FreeMASTER Serial Communication Driver

User Guide

Important Notice

Freescale provides the enclosed product(s) under the following conditions:

This evaluation kit is intended for use of ENGINEERING DEVELOPMENT OR EVALUATION PURPOSES ONLY. It is provided as a sample IC pre-soldered to a printed circuit board to make it easier to access inputs, outputs, and supply terminals. This EVB may be used with any development system or other source of I/O signals by simply connecting it to the host MCU or computer board via off-the-shelf cables. This EVB is not a Reference Design and is not intended to represent a final design recommendation for any particular application. Final device in an application will be heavily dependent on proper printed circuit board layout and heat sinking design as well as attention to supply filtering, transient suppression, and I/O signal quality.

The goods provided may not be complete in terms of required design, marketing, and or manufacturing related protective considerations, including product safety measures typically found in the end product incorporating the goods. Due to the open construction of the product, it is the user's responsibility to take any and all appropriate precautions with regard to electrostatic discharge. In order to minimize risks associated with the customers applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards. For any safety concerns, contact Freescale sales and technical support services.

As a prototype, this product does not fall within the scope of the European Union directive on electromagnetic compatibility and therefore may not meet the technical requirements of the directive. Please be aware that the products received may not be regulatory compliant or agency certified (FCC, UL, CE, etc.).

Should this evaluation kit not meet the specifications indicated in the kit, it may be returned within 30 days from the date of delivery and will be replaced by a new kit.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typical”, must be validated for each customer application by customer’s technical experts.

Freescale does not convey any license under its patent rights nor the rights of others. Freescale products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale product could create a situation where personal injury or death may occur.

Should a Buyer purchase or use Freescale products for any such unintended or unauthorized application, The Buyer shall indemnify and hold Freescale and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale was negligent regarding the design or manufacture of the part.

Table of Contents

Paragraph Number | Page Number

Important Notice

Chapter 1 INTRODUCTION

1.1 The Software 1
1.2 Replacing Existing Drivers 1
1.3 Quick_Start Tools 2
1.4 License 2

Chapter 2 DESCRIPTION

2.1 Introduction 5
2.2 Features 5
2.2.1 Board Detection 5
2.2.2 Memory Read 5
2.2.3 Memory Write 6
2.2.4 Masked Memory Write 6
2.2.5 Oscilloscope 6
2.2.6 Recorder 6
2.2.7 Target-side Addressing (TSA) 6
2.2.8 TSA Safety 6
2.2.9 Application Commands 6
2.3 Driver Files 7
2.4 Driver Configuration 7
2.4.1 Quick_Start Graphical Configuration Tool 8
2.4.2 Configurable Items 8
2.4.3 Driver Interrupt Modes 10
2.5 Data Types 11
2.6 SCI Initialization 12
2.7 FreeMASTER Recorder Calls 12
2.8 Driver Usage 12

Chapter 3 DRIVER API

3.1 Control API 13
3.1.1 FMSTR_Init 13
3.1.2 FMSTR_Poll 13
3.1.3 FMSTR_Isr 13
3.2 Recorder API 14
3.2.1 FMSTR_Recorder 14
3.2.2 FMSTR_TriggerRec 14
3.2.3 FMSTR_SetUpRecBuff 15
3.3 Target-side Addressing API 15
3.3.1 TSA Table Definition 15

3.3.2 TSA Table List 16
3.4 Application Commands API 17
3.4.1 FMSTR_GetAppCmd 17
3.4.2 FMSTR_GetAppCmdData 17
3.4.3 FMSTR_AppCmdAck 17
3.4.4 FMSTR_AppCmdSetResponseData 18
3.4.5 FMSTR_RegisterAppCmdCall 18
3.5 API Data Types 19

Chapter 4 PLATFORM-SPECIFIC TOPICS

4.1 Platform-dependent Code 21
4.2 DSP56F8xx Digital Signal Processors 22
4.2.1 56F8xx-specific Driver Files 22
4.2.2 56F8xx-specific Configuration Options 22
4.3 56F8xxx Digital Signal Controllers 23
4.3.1 56F8xxx-specific Driver Files 23
4.3.2 56F8xx-specific Configuration Options 23
4.4 HC08 / HCS08 Microcontrollers 24
4.4.1 HC08-specific Driver Files 24
4.4.2 HC08-specific Configuration Options 24
4.5 HC12 / HCS12 / HCS12X Microcontrollers 25
4.5.1 HC12-specific Driver Files 25
4.5.2 HC12-specific Configuration Options 25
4.6 MPC5xx and MPC55xx PowerPC Processors 26
4.6.1 MPC55xx-specific Driver Files 26
4.6.2 MPC5xx-specific Driver Files 26
4.6.3 MPC55xx-specific Configuration Options 26
4.7 MCF52xx ColdFire Processors 27
4.7.1 MCF52xx-specific Driver Files 27
4.7.2 MCF52xx-specific Configuration Options 27

Appendix A References 28

Appendix B Revision History 29

List of Tables

Table Number		Page Number
Table 1-1.	Supported Platforms	1
Table 2-1.	Driver Configuration Options	8
Table 2-2.	Driver Interrupt Modes	11
Table 3-1.	TSA Type Constants	16
Table 3-2.	Public Data Types	19
Table 3-3.	TSA Public Data Types	20
Table 3-4.	Private Data Types	20
Table 4-1.	56F8xxx Platform	23
Table 4-2.	HC08 Platform	24
Table 4-3.	HC12 Platform	25

Chapter 1 INTRODUCTION

FreeMASTER is a PC-based application serving as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on Freescale Semiconductor processing units. This document describes the embedded-side software driver which implements the serial interface between the application and the host PC. The serial interface covers the UART SCI communication for all supported devices and the EOnCE/JTAG communication for 56F8xxx family of hybrid microcontrollers.

This document does not describe the other communication drivers supported by the FreeMASTER tool, like CAN, USB or Ethernet. Please see the respective documentation for these drivers.

1.1 The Software

At the time of writing this document, the software driver supports the Freescale products as summarized in Table 1-1 below. In the future, more platforms may be expected to be supported by the driver, however the approach to configuration and usage of the driver should remain compatible with the one described herein. Please see the release notes of the latest driver installation pack for an up-to-date list of supported devices, as well as respective addendum to this User Manual.

Table 1-1. Supported Platforms

CPU / Platform	Tested with Compiler	Sample Applications Available
DSP56F800 Digital Signal Processors	CodeWarrior 56800/E Hybrid Controllers version 7.2	Standalone application created using the DSP56F800_Quick_Start r2.0 Board: DSP56F803EVM
56F8xxx Digital Signal Controllers (Hybrid Controllers)	CodeWarrior 56800/E Hybrid Controllers version 7.2	Standalone applications created using the DSP56F800E_Quick_Start r2.1 Boards: 56F8346EVM, DEMO56F8013
HC08 / HCS08 Microcontrollers	CodeWarrior Development Studio for Motorola HC08 3.1	Applications created from original CodeWarrior project stationery Devices: MC68HC908KX8, MC9S08RG6, MC9S08QG8, MC9S08GB60/GT60
HC12 / HCS12 Microcontrollers	CodeWarrior Development Studio for Motorola HC12 3.1	Applications created from original CodeWarrior project stationery Device: MC9S12DP256
HCS12X Microcontrollers	CodeWarrior Development Studio for Motorola HCS12X 4.1	Applications created from original CodeWarrior project stationery Device: MC9S12XDP512 (both banked and large data models)
MPC55xx PowerPC Processors	CodeWarrior Development Studio for MPC55xx V1.2	Standalone application created using the MPC5500_Quick_Start r0.4 Board: MPC5553DEMO
MPC5xx PowerPC Processors	CodeWarrior EPPC 6.6 and CodeWarrior Development Studio for MPC5xx V8.1	Standalone application created using the MPC500_Quick_Start r4.0 Boards: CME-0555, MPC556EVB
MCF52xx ColdFire Processors	CodeWarrior Development Studio for ColdFire Architectures, version 6.1	Application created using the built-in CodeWarrior project-creation wizard Boards: M5213EVB,M5235EVB,M5282LITE

1.2 Replacing Existing Drivers

The driver described in this document replaces the older drivers which were available separately for individual platforms, and were known as “PC Master” SCI drivers. As the FreeMASTER tool remains fully compatible with the communication interface provided by the old PC Master drivers, it is strongly recommended to upgrade existing embedded applications to this new FreeMASTER driver.

The main advantage of the new driver is a unification across all supported Freescale processor products, as well as several new features that were added. One of the key features implemented in the new driver is a “Target-side Addressing” (TSA), which enables an embedded application to describe memory objects it grants the host an access to. By enabling the so-called “TSA-Safety” option, the application memory can be protected from illegal or invalid memory accesses.

1.3 Quick_Start Tools

The Freescale Quick_Start tools available for MPC500/5500, DSP56F800 and 56F800E platforms also include some (older) versions of the FreeMASTER Serial Communication Driver. The standalone driver code described in this document is based on these individual Quick_Start drivers, and may be used as their up-to-date replacement. The FreeMASTER Serial Communication Driver installer will offer an automatic update of such FreeMASTER and PC_Master drivers, in all Quick_Start tools which exist on the host computer.

1.4 License

The license conditions of both the FreeMASTER tool and the embedded-side software drivers restrict the usage to embedded applications utilizing the processor units from Freescale Semiconductor only. The full text of the license conditions follows:

FREESCALE SEMICONDUCTOR SOFTWARE LICENSE AGREEMENT

[SOFTWARE FOR: FreeMASTER Serial Communication Driver]

This is a legal agreement between you (either as an individual or as an authorized representative of your employer) and Freescale Semiconductor, Inc. (“Freescale”). It concerns your rights to use this file and any accompanying written materials produced by Freescale (the “Software”). In consideration for Freescale allowing you to access the Software, you are agreeing to be bound by the terms of this Agreement. If you do not agree to all of the terms of this Agreement, do not install or download the Software. If you change your mind later, stop using the Software and delete all copies of the Software in your possession or control. Any copies of the Software that you have already distributed, where permitted, and do not destroy will continue to be governed by this Agreement. Your prior use will also continue to be governed by this Agreement. Please note that 3rd party products, including but not limited to software (“3rd Party Products”), may be distributed in conjunction with the Software. This Agreement does not apply to those 3rd Party Products, which will be subject to their own licensing terms.

LICENSE GRANTS. Your license to the Software and applicable restrictions vary depending on the nature of the Software provided. Review the following grants carefully to ensure your compliance.

IF SOFTWARE PROVIDED IN SOURCE FORM. Freescale grants to you the non-exclusive, non-transferable right (1) to use the Software exclusively in conjunction with a development platform from Freescale or other development, prototype, or production platform utilizing at least one processor from Freescale (“Exclusive Use”), (2) to reproduce the Software as necessary to accomplish the Exclusive Use, (3) to prepare derivative works of the Software as necessary to accomplish the Exclusive Use, (4) to distribute the Software and derivative works thereof in object (machine-readable) form only as integrated with a development platform from Freescale or other development, prototype, or production platform utilizing at least one processor from Freescale, and (5) to sublicense to others the right to use the distributed Software. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

IF SOFTWARE PROVIDED IN OBJECT FORM ONLY. Freescale grants to you the non-exclusive, non-transferable right (1) to use the Software exclusively in conjunction with a development platform from Freescale or other development, prototype, or production platform utilizing at least one processor from Freescale (“Exclusive Use”), (2) to reproduce the Software as necessary to accomplish the Exclusive Use, (3) to distribute the Software only as integrated with a development platform from Freescale or other development, prototype, or production platform utilizing at least one processor from Freescale, and (4) to sublicense to others the right to use the distributed Software.

The Software is provided to you only in object (machine-readable) form. You may exercise the rights above only with respect to such object form. You may not translate, reverse engineer, decompile, or disassemble the Software except to the extent applicable law specifically prohibits such restriction. In addition, you must prohibit your sublicensees from doing the same. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

FOR TOOLS. Freescale grants to you the non-exclusive, non-transferable right (1) to use the Software exclusively in conjunction with a development platform from Freescale (“Exclusive Use”), and (2) to reproduce the Software. The Software is provided to you only in object (machine-readable) form. You may not distribute or sublicense the Software to others. You may exercise the rights above only with respect to such object form. You may not translate, reverse engineer, decompile, or disassemble the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

COPYRIGHT. The Software is licensed to you, not sold. Freescale owns the Software, and United States copyright laws and international treaty provisions protect the Software. Therefore, you must treat the Software like any other copyrighted material (e.g., a book or musical recording). You may not use or copy the Software for any other purpose than what is described in this Agreement. Except as expressly provided herein, Freescale does not grant to you any express or implied rights under any Freescale or third party patents, copyrights, trademarks, or trade secrets. Additionally, you must reproduce and apply any copyright or other proprietary rights notices included on or embedded in the Software to any copies or derivative works made thereof, in whole or in part, if any.

SUPPORT. Freescale is NOT obligated to provide any support, upgrades or new releases of the Software. If you wish, you may contact Freescale and report problems and provide suggestions regarding the Software. Freescale has no obligation whatsoever to respond in any way to such a problem report or suggestion. Freescale may make changes to the Software at any time, without any obligation to notify or provide updated versions of the Software to you.

LIMITED WARRANTY ON MEDIA. Freescale warrants that the media on which the Software is recorded will be free from defects in materials and workmanship under normal use for a period of 90 days from the date of purchase as evidenced by a copy of the receipt.

Freescale's entire liability and your exclusive remedy under this warranty will be replacement of the defective media returned to Freescale with a copy of the receipt. Freescale will have no responsibility to replace any media damaged by accident, abuse or misapplication. This warranty extends only to you and may be invoked only by you for your customers. Freescale will not accept warranty returns from your customers.

NO ADDITIONAL WARRANTY. EXCEPT FOR THE LIMITED WARRANTY ON MEDIA PROVIDED ABOVE, THE SOFTWARE AND 3RD PARTY PRODUCTS, IF ANY, ARE PROVIDED “AS IS”. YOUR USE OF THE SOFTWARE OR 3RD PARTY PRODUCTS IS AT YOUR SOLE RISK. SHOULD THE SOFTWARE OR ANY 3RD PARTY PRODUCT PROVE DEFECTIVE, YOU (AND NOT FREESCALE OR ANY FREESCALE REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. FREESCALE EXPRESSLY DISCLAIMS ALL WARRANTIES WITH RESPECT TO THE SOFTWARE AND 3RD PARTY PRODUCTS, WHETHER SUCH WARRANTIES ARE EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. YOU EXPRESSLY ASSUME ALL LIABILITIES AND RISKS, FOR ANYONE'S USE OR OPERATION OF ANY APPLICATION PROGRAMS YOU MAY CREATE WITH THE SOFTWARE.

INDEMNITY. Freescale will defend, at its expense, any suits asserted against you based upon a claim that the Software as provided by Freescale infringes a U.S. patent or copyright or misappropriates a trade secret, and pay costs and damages finally awarded based upon such suit, if you: (1) promptly notify Freescale in writing as soon as reasonably practicable after you first become aware of the claim of infringement or misappropriation, but in no event later than 15 days of the date on which you first received notice of the claim; and (2) at Freescale's request and expense, give Freescale sole control of the suit and all requested assistance for defense of the suit. Freescale will not be liable for any settlement made without its written consent. If the use or sale of any Software component program licensed under this Agreement is enjoined as a result of such suit, Freescale at its option and at no expense to you, will: (1) obtain for you the right to use such program consistent with the license granted in this Agreement for the affected program; (2) substitute an equivalent program and extend this indemnity thereto; or (3) accept the return of the program and refund the portion of the license fee for such component program less reasonable charge for prior use. If an infringement or misappropriation claim related to the Software is alleged prior to completion of delivery, Freescale has the right to decline to make further shipments notwithstanding any other provision of this Agreement. This indemnity does not extend to any suit based upon any infringement or alleged infringement arising from any program furnished by Freescale that is: (1) altered in any way by you or any third party if the alleged infringement would not have occurred but for such alteration; (2) combined with any other products or elements not furnished by Freescale if the alleged infringement would not have occurred but for such combination; (3) designed or manufactured in accordance with your designs, specifications or instructions if the alleged infringement would not have occurred but for such designs, specifications or instructions; or (4) designed or manufactured in compliance with standards issued by any public or private standards body if the alleged infringement would not have occurred but for compliance with such standards. In no event will Freescale indemnify you or be liable in any way for royalties payable based on a per use basis, or any royalty basis other than a reasonable royalty based upon revenue derived by Freescale from your license of the Software.

THE INDEMNITY PROVIDED IN THIS SECTION IS THE SOLE, EXCLUSIVE, AND ENTIRE LIABILITY OF FREESCALE AND THE REMEDIES PROVIDED IN THIS SECTION SHALL BE YOUR EXCLUSIVE REMEDIES AGAINST FREESCALE FOR PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION AND IS PROVIDED IN LIEU OF ALL WARRANTIES, EXPRESS, IMPLIED OR STATUTORY IN REGARD THERETO, INCLUDING, WITHOUT LIMITATION, THE WARRANTY AGAINST INFRINGEMENT SPECIFIED IN THE UNIFORM COMMERCIAL CODE.

LIMITATION OF LIABILITY. IN NO EVENT WILL FREESCALE BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY INCIDENTAL, SPECIAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, OR LOST PROFITS, SAVINGS, OR REVENUES TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW.

COMPLIANCE WITH LAWS; EXPORT RESTRICTIONS. You must use the Software in accordance with all applicable U.S. laws, regulations and statutes. You agree that neither you nor your licensees (if any) intend to or will, directly or indirectly, export or transmit the Software to any country in violation of U.S. export restrictions.

GOVERNMENT USE. Use of the Software and any corresponding documentation, if any, is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(l) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Freescale Semiconductor, Inc., 6501 William Cannon Drive West, Austin, TX, 78735.

HIGH RISK ACTIVITIES. You acknowledge that the Software is not fault tolerant and is not designed, manufactured or intended by Freescale for incorporation into products intended for use or resale in on-line control equipment in hazardous, dangerous to life or potentially life-threatening environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems.

CHOICE OF LAW; VENUE; LIMITATIONS. You agree that the statutes and laws of the United States and the State of Texas, USA, without regard to conflicts of laws principles, will apply to all matters relating to this Agreement or the Software, and you agree that any litigation will be subject to the exclusive jurisdiction of the state or federal courts in Texas, USA. You agree that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to this Agreement or the Software must be filed within one (1) year after such claim or cause of action arose or be forever barred.

PRODUCT LABELING. You are not authorized to use any Freescale trademarks, brand names, or logos.

ENTIRE AGREEMENT. This Agreement constitutes the entire agreement between you and Freescale regarding the subject matter of this Agreement, and supersedes all prior communications, negotiations, understandings, agreements or representations, either written or oral, if any. This Agreement may only be amended in written form, executed by you and Freescale.

SEVERABILITY. If any provision of this Agreement is held for any reason to be invalid or unenforceable, then the remaining provisions of this Agreement will be unimpaired and, unless a modification or replacement of the invalid or unenforceable provision is further held to deprive you or Freescale of a material benefit, in which case the Agreement will immediately terminate, the invalid or unenforceable provision will be replaced with a provision that is valid and enforceable and that comes closest to the intention underlying the invalid or unenforceable provision.

NO WAIVER. The waiver by Freescale of any breach of any provision of this Agreement will not operate or be construed as a waiver of any other or a subsequent breach of the same or a different provision.

Chapter 2 DESCRIPTION

2.1 Introduction

This section describes how to add the FreeMASTER Serial Communication Driver into your application, how to configure, and how to enable a connection to the FreeMASTER visualization tool.

2.2 Features

The FreeMASTER driver implements all the features necessary to establish a communication with the FreeMASTER visualization tool. The driver is a newly rewritten (backward compatible) version of the older “PC Master” driver, adding the support for new platforms, better code structure and readability, and also new features like Target-side Addressing (TSA), TSA-Safety, and Application Command Callback functions.

FreeMASTER protocol features:

- Read/Write Access to any memory location on the target.
- Atomic bit manipulation on target memory (bit-wise write access).
- Oscilloscope access - optimized real time access to variables (up to 8 variables). Sample rate depends on the communication speed.
- Recorder - access to fast transient recorder running on-board as a part of FreeMASTER driver. Sample rate is limited by microcontroller CPU speed only. The length of data recorded depends on amount of available memory, 64kB maximum.
- Application commands - high level message delivery from PC to the application.

FreeMASTER driver features:

- Full FreeMASTER protocol implementation
- SCI or EOnCE/JTAG as a native communication interface
- Ability to write-protect memory regions or individual variables
- Ability to deny access to unsafe memory
- Two ways to handle Application Commands
  - Classic: the application polls the App.Command status to determine any command is pending.
  - Callback: the application registers a callback function which is automatically invoked upon reception of a given command.
  - The two approaches may be mixed in the application. Callback commands do not appear in the polling mechanism.

This following sections describe briefly all FreeMASTER features implemented by the driver. Please see also the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune or control your embedded application.

2.2.1 Board Detection

FreeMASTER protocol defines a standard way the host PC reads the platform-specific information it needs to access target resources. The board information includes the following parameters:

- the version of the driver and version of the protocol implemented
- driver name *
- target processor byte ordering (little/big endian)
- communication buffer length
- address space granularity (1 byte on most platforms, 2 on 56F8xx DSP)
- recorder capabilities *

On smaller devices with limited Flash and RAM memory resources, the Board Information may be restricted to a “brief” version by excluding unnecessary items (the items marked by * in the list above).

2.2.2 Memory Read

This basic feature enables the host PC to read any data memory location by specifying an address and size of the required memory area. The device response frame should fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during Board Detection, and splits the large-block request to multiple partial requests.

A slightly optimized variant of “Memory Read” protocol feature is a “Variable Read” feature, which enables to read 1, 2 or 4 byte variables while saving one byte on the communication line.

2.2.3 Memory Write

Similarly as the Memory Read operation, the Memory Write feature enables to write any RAM memory location on the target device. Again, a single write command frame should fit onto the targets communication buffer, which is needed to split a large write requests into a smaller ones.

A slightly optimized “Variable Write” variants exist to write 2 and 4 byte variables.

2.2.4 Masked Memory Write

To implement a write access to single bits or group of bits of a target variables, the “Masked Memory Write” feature is available in the FreeMASTER protocol. Except the AND-mask is applied to the data values being written, this feature is similar to classic “Memory Write”.

A slightly optimized “Masked Variable Write” variants exist to write 1and 2 byte variables.

2.2.5 Oscilloscope

The protocol enables up to eight variables to be selected to be read at once on a single request from the host. This feature is called an “Oscilloscope” and the FreeMASTER tool uses it to display real-time graph of variable values.

This is an optional feature and if disabled, the FreeMASTER uses the standard “Memory Read” or “Variable Read” accesses to read the graphed variables sequentially.

2.2.6 Recorder

The protocol defines a standard way of how to select up to eight variables on the target whose values are periodically recorded into dedicated on-board memory buffer. After the data sampling is stopped, either on a host request or by evaluating a threshold-crossing condition, the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of communication line, so it enables to display the variable transitions in a very high resolution.

This is an optional feature and may be disabled if Recorder is not needed in the application.

2.2.7 Target-side Addressing (TSA)

With this feature, the user is able to describe the variables and structure data types directly in the application source code and make this information available for the FreeMASTER tool. The tool may then use this information instead of reading it from the application’s ELF/Dwarf executable file.

The Target-side Addressing enables the user to create so-called TSA-tables, and to put them directly into the embedded application. The TSA tables contain descriptors of variables the user wants to make visible to the host. The descriptors are capable of describing the memory areas by specifying an address and size of the block or more conveniently by using directly the C variable names. The user may also put a type information about structures, unions, or arrays into the TSA table.

2.2.8 TSA Safety

When TSA is enabled in the application, the TSA-Safety can be enabled and can make the memory accesses validated directly by the embedded-side driver. When the TSA-Safety is turned on, any memory request received from the host is validated and is accepted only if it falls to any TSA-described object. Moreover, the TSA entries can be declared as Read-Write or Read-Only so the driver can actively deny a write access to the Read-Only objects.

2.2.9 Application Commands

The Application Commands are high-level messages the host may deliver to the embedded application for further processing. The embedded application may either poll the status, or can be called back when a new Application Command arrives to be processed. After processing, when embedded application “acknowledges” the command is handled, the host receives back the Result Code. Both the Application Commands and the Result Codes are specific to a given application and it is a programmer’s responsibility to define them. The FreeMASTER protocol and FreeMASTER driver only implements the delivery channel and a set of API calls to enable Application Command processing.

2.3 Driver Files

After installing the software, the driver source files can be found in the “driver” subdirectory, further divided into the sub-directories.

- **`src_platforms / platform-specific directory`** - one directory exists for each supported processor platform (56F8xx, 56F8xxx, HC08, HC12, MPC55xx, etc.). There is a master header file freemaster.h plus platform-dependent C and header files. The C file needs to be added to the project for compilation and linking. The freemaster.h file should be included in your application, wherever you need to call any of FreeMASTER driver API functions.
  - **`freemaster.h`** - master driver header file. Declares the common data types, macros and prototypes of the FreeMASTER driver API functions.
  - **`freemaster_XXX.c`** (XXX stands for platform identifier) - this file contains the platform-specific functions to access data memory, communication buffer memory, serial interface interrupts, and other platform-specific code.
  - **`freemaster_XXX.h`** (XXX stands for platform identifier) - this file defines the platform-specific SCI access macros, memory access inline functions, and other platform-specific declarations.
  - **`freemaster_cfg.h.example`** - this file may serve as an example of FreeMASTER driver configuration file. Save this file into your project source code directory and rename it to “**`freemaster_cfg.h`**”. FreeMASTER driver code includes this file to get your project-specific configuration options, and to optimize the compilation of the driver.
- **`src_common` directory** - contains common driver source files, shared by the driver for all supported platforms. All the .c files should be added to your project, compiled and linked together with your application.
  - **`freemaster_serial.c`** - implements the serial communication buffers, FIFO queuing, and other communication-related operations. This file uses a SCI or JTAG module access macros from the platform-dependent header file (see above).
  - **`freemaster_protocol.c`** - implements the FreeMASTER protocol decoder and handles the basic memory read or memory write commands.
  - **`freemaster_rec.c`** - handles the recorder-specific commands and implements the recorder sampling routine. In case the recorder is disabled by the FreeMASTER driver configuration file, this file compiles to empty API functions only.
  - **`freemaster_scope.c`** - handles the oscilloscope-specific commands. In case the oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles void.
  - **`freemaster_appcmd.c`** - handles the communication commands used to deliver and execute so-called “Application Commands” within the context of the embedded application. See the FreeMASTER Tool User Manual for more details on this feature. In case the Application Commands are disabled by the FreeMASTER driver configuration file, this file compiles to empty API functions only.
  - **`freemaster_tsa.c`** - handles the commands specific to Target-Side Addressing feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. In case the TSA is disabled by the FreeMASTER driver configuration file, this file compiles void.

The header files from the **`src_common`** directory are private to the driver and should not be included anywhere in your application. Similarly as with the platform-dependent directory, you should add the `src_common` to your compiler’s “include” search path.

- **`freemaster_private.h`** - contains declarations of functions and data types used internally in the driver. Based on the platform identification macro declared in `freemaster.h`, this file includes a platform-dependent header file (`freemaster_`*`XXX`*`.h`). The `freemaster_private.h` file also contains C pre-processor statements to perform intensive compile-time verification of the driver configuration.
- `freemaster_protocol.h` - defines FreeMASTER protocol constants used internally by the driver
- `freemaster_tsa.h` - this file contains declaration of the macros used to define the TSA memory descriptors. This file is indirectly included to the user application code (from `freemaster.h`).

2.4 Driver Configuration

The driver is configured using a single header file, named **`freemaster_cfg.h`**. The user creates this file and saves it together with his other project source files. The FreeMASTER driver source files include the `freemaster_cfg.h` file, and use macros defined here for conditional and parametrized compilation. The AC compiler must be able to locate the configuration file when compiling the driver files. Typically, this can be achieved by putting this file into a directory where the other project-specific included files are stored.

As a starting point to create the configuration file, it is possible to get the **`freemaster_cfg.h.example`** file from the platform-specific source directory, rename it to `freemaster_cfg.h,` and save it into the project area.

NOTE

It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code directory. The configuration file should be placed to project-specific location, so it does not affect other applications which make use of the driver.

2.4.1 Quick_Start Graphical Configuration Tool

As described in the Section 1.4, "License on page 1-2, the FreeMASTER Serial Communication Driver may be incorporated into existing Freescale Quick_Start source files (MPC500/5500 and 56F800/E platforms).

When using the FreeMASTER driver in an application based on the Quick_Start project, the `freemaster_cfg.h.quickstart` may be renamed to `freemaster_cfg.h` and it can be used as a configuration file placeholder. The `freemaster_cfg.h.quickstart` file only includes the `appconfig.h` file, and other Quick_Start-specific header files which enable the FreeMASTER driver to be configured using the Quick_Start Graphical Configuration Tool.

Working in the Quick_Start environment and using its Graphical Configuration Tool helps the user to easily configure all processor peripheral modules in a user-friendly graphical application. This includes also a configuration of system clocks and the SCI module required by the FreeMASTER driver, as well as the FreeMASTER driver itself. All configurable items described in Section 2.4.2, "Configurable Items are graphically configurable.

2.4.2 Configurable Items

The Table 2-1 below describes the `freemaster_cfg.h` configuration options which are common to all supported platforms. See Chapter 4, "PLATFORM-SPECIFIC TOPICS beginning on page 4-21 for a detailed descriptions of the platform-specific options.

Table 2-1. Driver Configuration Options

Statement	Values	Description
#define FMSTR_LONG_INTR #define FMSTR_SHORT_INTR #define FMSTR_POLL_DRIVEN	boolean (0 or 1) boolean (0 or 1) boolean (0 or 1)	Exactly one of the three macros must be defined non-zero, others must be defined zero. The non-zero-defined constant selects the interrupt mode of the driver. See Section 2.4.3, "Driver Interrupt Modes. FMSTR_LONG_INTR - long interrupt mode FMSTR_SHORT_INTR - short interrupt mode FMSTR_POLL_DRIVEN - poll driven mode
#define FMSTR_SCI_BASE	address	Specify the base address of the SCI peripheral module to be used for communication. If you use the symbolic name, make sure you also include the header file where the symbol is defined or declared.
#define FMSTR_COMM_BUFFER_SIZE	0...255	Specify the size of the communication buffer to be allocated by the driver. When undefined, or defined as 0 (recommended) the buffer size will be automatically calculated in freemaster_private.h, based on the driver features selected. Default: 0 (automatic)
#define FMSTR_COMM_RQUEUE_SIZE	0...255	Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode. Default: 32 bytes
#define FMSTR_USE_READMEM	boolean (0 or 1)	Define as non-zero to implement the "Memory Read" command. Recommended. Default: "true".
#define FMSTR_USE_READVAR	boolean (0 or 1)	Define as non-zero to implement the "Variable Read" command. Default "false".

Table 2-1. Driver Configuration Options

Statement	Values	Description
#define FMSTR_USE_WRITEMEM	boolean (0 or 1)	Define as non-zero to implement the “Memory Write” command. Recommended. Default: “true”.
#define FMSTR_USE_WRITEVAR	boolean (0 or 1)	Define as non-zero to implement the “Variable Write” command. Default “false”.
#define FMSTR_USE_WRITEMEMMASK	boolean (0 or 1)	Define as non-zero to implement the “Masked Memory Write” command. Recommended. Default: “true”.
#define FMSTR_USE_WRITEVARMASK	boolean (0 or 1)	Define as non-zero to implement the “Masked Variable Write” command. Default “false”.
Oscilloscope		
#define FMSTR_USE_SCOPE	boolean (0 or 1)	Define as non-zero to implement the “Oscilloscope” feature. Default “false”.
#define FMSTR_MAX_SCOPE_VARS	2...8	Number of variables to be supported in Oscilloscope.
Recorder		
#define FMSTR_USE_RECORDER	boolean (0 or 1)	Define as non-zero to implement the “Recorder” feature. Default “false”.
#define FMSTR_MAX_REC_VARS	2...8	Number of variables to be supported in Recorder.
#define FMSTR_REC_OWNBUFF	boolean (0 or 1)	When true (non-zero), the driver does NOT allocate the recorder buffer. The user must call the `FMSTR_SetUpRecBuff` function to configure the recorder buffer before using it. Default: “false” (driver allocates the buffer)
#define FMSTR_REC_BUFF_SIZE	16bit number	When FMSTR_REC_OWNBUFF is “false”, this constant defines the size of the recorder buffer to be allocated by the driver.
#define FMSTR_REC_TIMEBASE	16bit number	Ths constant tells the host how often you call the recorder sampling routine (`FMSTR_Recorder`) in your application. The host uses this information to draw an x-axis of the recorder graph properly. Use 0 as “unknown” (the x-axis will be drawn indexed only). Use one of the following macros to specify the time value (x is a14bit constant): FMSTR_REC_BASE_SECONDS(x) FMSTR_REC_BASE_MILLISEC(x) FMSTR_REC_BASE_MICROSEC(x) FMSTR_REC_BASE_NANOSEC(x)
Application Commands		
#define FMSTR_USE_APPCMD	boolean (0 or 1)	Define as non-zero to implement the “Application Commands” feature. Default “false”.

Table 2-1. Driver Configuration Options

Statement	Values	Description
#define FMSTR_APPCMD_BUFF_SIZE	1...255	Size of the "Application Command" data buffer allocated by the driver. The buffer stores the (optional) parameters of "Application Command" which waits to be processed.
#define FMSTR_MAX_APPCMD_CALLS	0...255	Number of different "Application Commands" you will be able to assign to a callback handler function (see `FMSTR_RegisterAppCmdCall`). Default: 0
Target-side Addressing		
#define FMSTR_USE_TSA	boolean (0 or 1)	Define as non-zero to implement the "Target-Side Addressing" feature. Default "false".
#define FMSTR_USE_TSA_SAFETY	boolean (0 or 1)	Enable memory access validation in the FreeMASTER driver. The host will not be able to access the memory not described by any TSA descriptor. A write access will be denied to "Read-Only" objects.
#define FMSTR_USE_TSA_INROM	boolean (0 or 1)	Declare all TSA descriptors as "const", which enable the linker to put the data into Flash memory. The actual result depends on your linker settings or linker commands used in your project. Default: "false"
Advanced Options		
#define FMSTR_USE_NOEX_CMDS	boolean (0 or 1)	Enable 16-bit addressing commands in the FreeMASTER driver. The FreeMASTER protocol commands carrying the 16-bit addresses are also called "standard" or "non-EX". See the note in the parameter below.
#define FMSTR_USE_EX_CMDS	boolean (0 or 1)	Enable "EX" FreeMASTER protocol commands to be processed by the driver. "EX" commands only differ from the "standard" commands by using 32-bit addresses. "EX" commands are always two bytes longer than its standard counterparts. **Note:** Although being configurable items, it is not really recommended to specify these two parameters in the configuration file. The default values of both options are defined in the platform-dependent header file and it should not be necessary to override them manually. See Chapter 4, "PLATFORM-SPECIFIC TOPICS for more details about individual platforms.

2.4.3 Driver Interrupt Modes

To implement a serial communication, the FreeMASTER driver handles the SCI module receive and transmit requests. The user selects whether the driver processes the SCI communication automatically as an interrupt service routine, or if it should only poll the status of the module, typically during the application idle time. See table Table 2-2 for a description of each mode.

Table 2-2. Driver Interrupt Modes

Mode	Description
Completely Interrupt-Driven (FMSTR_LONG_INTR = 1)	Both the SCI communication and the FreeMASTER protocol decoding and execution is done in the `FMSTR_Isr` interrupt service routine. As the protocol execution may be a lengthy task (especially with TSA-Safety enabled), it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application, and if the FreeMASTER interrupt is assigned to a lower (the lowest) priority. The application should subscribe the `FMSTR_Isr` function as the SCI interrupt vector (or multiple vectors in case the SCI receive, transmit and/or communication errors are handled by different interrupts).
Mixed Interrupt and Polling Modes (FMSTR_SHORT_INTR = 1)	The raw SCI communication is handled by the `FMSTR_Isr` interrupt service routine, while the protocol decoding and execution is handled in the `FMSTR_Poll` routine. The user typically calls the `FMSTR_Poll` during the idle time in the application “main loop”. The interrupt processing is relatively fast and deterministic. On a SCI receive event, the received character is only placed into a FIFO-like queue and is not further processed. When transmitting, the characters are just fetched from the prepared transmit buffer. The application should subscribe the `FMSTR_Isr` function as the SCI interrupt vector (or multiple vectors in case the SCI receive, transmit and/or communication errors are handled by different interrupts). The user must assure the `FMSTR_Poll` function is called at least once per N “character time” periods. Where N is the length of the FreeMASTER FIFO queue (FMSTR_COMM_RQUEUE_SIZE) and “character time” is the time needed to transmit or receive a single byte over the SCI line.
Completely Poll-driven (FMSTR_POLL_DRIVEN = 1)	Both the SCI communication and the FreeMASTER protocol execution is done in the `FMSTR_Poll` routine. No interrupts are needed, the `FMSTR_Isr` code compiles to an empty function. When using this mode, the user must assure the `FMSTR_Poll` function is called by an application at least once per “SCI character time”, which is the time needed to transmit or receive a single character.

Be aware that in two latter modes (FMSTR_SHORT_INTR, FMSTR_POLL_DRIVEN), the protocol handling takes place in the `FMSTR_Poll` routine. An application interrupt may occur in the middle of “Read Memory” or “Write Memory” command execution, and may corrupt the variable being accessed by the FreeMASTER driver. In these two modes, it is not recommended to use the FreeMASTER tool to visualize or monitor the “volatile” variables, i.e. those being modified anywhere in the user interrupt code.

The same restriction applies even in the full interrupt mode (FMSTR_LONG_INTR), if the “volatile” variables are modified in a interrupt code of priority higher than the priority of the SCI interrupt.

2.5 Data Types

An easy-portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately declared data types, and a vast majority of platform-dependent code is separated in the platform-dependent source files. Data types, used in the driver API are all defined in the “freemaster.h” master header file, with other private data types defined in a platform-specific header file.

To prevent the name conflicts with the symbols used in the user’s application, all data types, macros and functions are prefixed with the `FMSTR_` prefix. There are no global variables used in the driver. Private variables are all declared as “static” and are prefixed with either `fmstr_` or `pcm_` prefix.

2.6 SCI Initialization

The FreeMASTER driver does NOT perform any initialization or configuration of the SCI module it uses to communicate. It is the user's responsibility to configure the communication module before the FreeMASTER driver is initialized by the `FMSTR_Init` call.

The user needs to configure the SCI receive and transmit pins, serial communication baudrate, parity (no-parity), character length (8 bits), and number of stop bits (one) before initializing the FreeMASTER driver. For either long or short interrupt mode of the driver (see Section 2.4.3, "Driver Interrupt Modes), the interrupt controller should be configured and the `FMSTR_Isr` function should be set as the SCI interrupt service routine.

On many Freescale platforms, user friendly graphical tools exist and enable a convenient way to configure the processor peripheral modules.

NOTE

It is not necessary to enable or unmask the SCI interrupts (TIE, RIE bits), nor it is necessary to activate the SCI receiver or transmitter engines (TE, RE bits) before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines as required during the run-time.

2.7 FreeMASTER Recorder Calls

When the FreeMASTER recorder is used in the application (FMSTR_USE_RECORDER), the user has to call the `FMSTR_Recorder` function periodically, in places where a data recording should occur. A typical place to call the recorder routine is at a timer or PWM interrupt, but it can be anywhere the user wants the variable values to be sampled. The example applications provided together with a driver code make a `FMSTR_Recorder` call in the main application loop.

In applications where calls to the `FMSTR_Recorder` occur equidistantly, the user may use the FMSTR_REC_TIMEBASE macro to let the host application know the recorder sampling period. If used in this way, the FreeMASTER recorder displays the X-axis of the recorder graph properly recalculated into a time domain.

2.8 Driver Usage

The following steps are necessary to enable a basic FreeMASTER connectivity in the application:

- Configure the FreeMASTER driver by creating or editing the `freemaster_cfg.h` file.
- Include the **`freemaster.h`** file to any application source file which makes the FreeMASTER API calls.
- For the FMSTR_LONG_INTR or FMSTR_SHORT_INTR modes, route the SCI (or JTAG) interrupts to `FMSTR_Isr` function and set the **interrupt priority levels** if applicable.
- Initialize the SCI or JTAG module. Set the baudrate, parity and other parameters of the communication. Do not enable the SCI interrupts.
- Call the `FMSTR_Init` function.
- For the FMSTR_SHORT_INTR or FMSTR_POLL_DRIVEN modes, start calling the `FMSTR_Poll` API function periodically in the application.
- For the FMSTR_SHORT_INTR or FMSTR_LONG_INTR modes, enable the interrupts.

Chapter 3 DRIVER API

This section describes the driver Application Programmer's Interface (API) needed to initialize and use the FreeMASTER Serial Communication Driver.

3.1 Control API

There are three functions which are key to initialize and to use the driver.

3.1.1 FMSTR_Init

Prototype

```
void FMSTR_Init(void)
```

Declaration

```
freemaster.h
```

Implementation

```
freemaster_protocol.c
```

Description

This function initializes internal variables of the FreeMASTER driver and enables the communication interface (SCI or JTAG). This function does not change the configuration of the selected communication module, the module must be initialized before the `FMSTR_Init` function is called.

A call to this function must occur before a call to any other FreeMASTER driver API function.

3.1.2 FMSTR_Poll

Prototype

```
void FMSTR_Poll(void)
```

Declaration

```
freemaster.h
```

Implementation

```
freemaster_serial.c
```

Description

In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see Section 2.4.3, "Driver Interrupt Modes). In the poll-driven mode, this function also handles the SCI communication. Typically, the user calls the `FMSTR_Poll` during the "idle" time in the main application loop.

To prevent receive data overflow (loss), the user must assure the `FMSTR_Poll` function is called at least once per time calculated as

$$N * T_{char}$$

Where N is equal to a length of receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). The N is 1 for the poll-driven mode.

The T_{char} is "character time", which is a time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function compiles as an empty function and may still be called. It may be worthwhile calling this function regardless of the interrupt mode used in the application. Such an approach enables convenient switching among different modes, but only by changing the configuration macros in the freemaster_cfg.h file.

3.1.3 FMSTR_Isr

Prototype

```
void FMSTR_Isr(void)
```

Declaration

```
freemaster.h
```

Implementation

```
platform-dependent C file (src_platforms directory)
```

Description

This is the SCI interrupt service routine of the FreeMASTER driver. In long or short interrupt modes (see Section 2.4.3, "Driver Interrupt Modes), this function has to be set as a SCI interrupt vector. On platforms where SCI processing is split to multiple interrupts, this function should be set as a vector for each such interrupt.

Note: In completely poll-driven mode, this function is compiled as an empty function and does not need to be used.

3.2 Recorder API

3.2.1 FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(void)
```

Declaration

```
freemaster.h
```

Implementation

```
freemaster_rec.c
```

Description

This function takes one sample of variables being recorded using the FreeMASTER recorder. If the recorder is not active at the moment when `FMSTR_Recorder` is called, the function returns immediately. When the recorder is active, the values of variables being recorded are copied to the recorder buffer and the trigger condition is evaluated.

If the trigger condition is satisfied, the recorder enters the post-trigger mode, where it counts the follow-up samples (`FMSTR_Recorder` function calls) and de-activates the recorder when required post-trigger samples are sampled.

Typically, the `FMSTR_Recorder` function is called in the Timer or PWM interrupt service routine. This function can also be called in the application main loop (for test purposes).

3.2.2 FMSTR_TriggerRec

Prototype

```
void FMSTR_TriggerRec(void)
```

Declaration

```
freemaster.h
```

Implementation

```
freemaster_rec.c
```

Description

This function forces the recorder trigger condition to happen, which causes the recorder to be automatically de-activated after post-trigger samples are sampled. This function can be used in the application when it needs to have the trigger occurrence under its control.

3.2.3 FMSTR_SetUpRecBuff

Prototype

```
void FMSTR_SetUpRecBuff(FMSTR_ADDR nBuffAddr, FMSTR_SIZE nBuffSize)
```

Declaration

```
freemaster.h
```

Implementation

```
freemaster_rec.c
```

Arguments

nBuffAddr `(in) - a pointer to the memory to be used as a recorder buffer`

nBuffSize `(in) - a size of the memory buffer (64kB maximum)`

Description

This function can only be used when the FMSTR_REC_OWNBUFF configuration constant is set to a non-zero value. The user calls this function to “give” the data buffer he allocated to the FreeMASTER driver, which will use it as a recorder buffer.

Up to 64kB buffer may be used as a recorder buffer.

3.3 Target-side Addressing API

When the Target-side Addressing (TSA) is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), the user must define so-called TSA tables in the application. This section describes macros which need to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There should be always one TSA Table List defined which informs the FreeMASTER driver about active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The FreeMASTER user is then able to create FreeMASTER variables based on these symbols. The TSA is supported in FreeMASTER version 1.2.39 and higher.

3.3.1 TSA Table Definition

The TSA table describes the static or global variables, together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and enable the FreeMASTER user to access individual structure members of the variable.

The TSA table definition begins with FMSTR_TSA_TABLE_BEGIN macro:

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

Where the `table_id` is any valid C-language symbol identifying the table. There can be any number of TSA tables in the application.

After this opening macro, the TSA descriptors are placed using any of the macros below:

```
FMSTR_TSA_RW_VAR(name, type)                // read/write variable entry

FMSTR_TSA_RO_VAR(name, type)                // read-only variable entry

FMSTR_TSA_STRUCT(struct_name)               // structure type entry

FMSTR_TSA_MEMBER(struct_name, member_name, type) // structure member entry
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

The TSA descriptor macros accept the following parameters:

- **name** - variable name. The variable must be defined before the TSA descriptor references it.
- **type** - variable or member type. Only one of the pre-defined type constants may be used, as described in Table 3-1.
- **struct_name** - structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- **member_name** - structure member name, without the dot at the beginning. The parent structure name is specified as a separate parameter in the FMSTR_TSA_MEMBER descriptor.

Note: Structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: In order to write-protect variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), the TSA-Safety feature needs to be enabled in the configuration file.

Note: Despite of its name, the FMSTR_TSA_STRUCT macro may also be used to describe union data types.

Table 3-1. TSA Type Constants

CONSTANT	Description
FMSTR_TSA_UINT8 FMSTR_TSA_UINT16 FMSTR_TSA_UINT32 FMSTR_TSA_UINT64	1, 2, 4 or 8-byte unsigned integer type. Use it for both the standard C-language types like unsigned char, unsigned short or unsigned long and the user-defined integer types commonly used on different platforms like UWord8, UWord16, uint8_t, unit16_t, Byte, Word, LWord etc.
FMSTR_TSA_SINT8 FMSTR_TSA_SINT16 FMSTR_TSA_SINT32 FMSTR_TSA_SINT64	1, 2, 4 or 8-byte signed integer type. Use it for both the standard C-language types like char, short or long and the user-defined integer types like Word8, Word16 or Word32, sint8_t, sint16_t etc.
FMSTR_TSA_FRAC16 FMSTR_TSA_FRAC32	Fractional data types. Although these types are treated as integer types in C-language, it may be beneficial to describe them using these macros, so the FreeMASTER treats them properly.
FMSTR_TSA_UFRAC16 FMSTR_TSA_UFRAC32	Unsigned fractional data types. Although these types are treated as unsigned integer types in C-language, it may be worthwhile to describe them using these macros, so the FreeMASTER treats them properly.
FMSTR_TSA_FLOAT	4-byte standard IEEE floating point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating point type
FMSTR_TSA_USERTYPE(**name**)	Structure or union type. You must specify the type name as an argument.

3.3.2 TSA Table List

There **must** be exactly one TSA Table List in the application. The list contains one entry for each TSA table which is defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro:

```
FMSTR_TSA_TABLE_LIST_BEGIN()
```

and continues with TSA table entries for each table:

```
FMSTR_TSA_TABLE(table_id)

FMSTR_TSA_TABLE(table_id2)

FMSTR_TSA_TABLE(table_id3)

...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro

```
FMSTR_TSA_TABLE_LIST_END()
```

3.4 Application Commands API

3.4.1 FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void)
```

Declaration

`freemaster.h`

Implementation

`freemaster_appcmd.c`

Description

This function can be used to detect if any Application Command is waiting to be processed by the application. If no command is pending, this function returns *FMSTR_APPCMDRESULT_NOCMD* constant. Otherwise, this function returns a code of the Application Command, which needs to be processed. Use the `FMSTR_AppCmdAck` call to acknowledge the Application Command after it is processed, and then to return the appropriate result code to the host.

The `FMSTR_GetAppCmd` function does not report commands for which a callback handler function exists. If the `FMSTR_GetAppCmd` function is called when a callback-registered Command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

3.4.2 FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* pDataLen)
```

Declaration

`freemaster.h`

Implementation

`freemaster_appcmd.c`

Arguments

pDataLen (out) - a pointer to a variable which receives the length of the data available in the buffer. It may be NULL when this information is not needed.

Description

This function can be used to retrieve the Application Command data, once the application determines the Application Command is pending (see FMTR_GetAppCmd function above).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application *after* the command is processed by the `FMSTR_AppCmdAck` call, the user needs to copy the data out to a private buffer.

3.4.3 FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT nResultCode)
```

Declaration

`freemaster.h`

Implementation

`freemaster_appcmd.c`

Arguments

nResultCode (in) - the result code which is to be returned to the FreeMASTER tool

Description

This function is used when Application Command processing is finished in the application. The nResultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called, and before the next Application Command arrives, the return value of the `FMSTR_GetAppCmd` function is FMSTR_APPCMDRESULT_NOCMD.

3.4.4 FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(

          FMSTR_ADDR nResultDataAddr,
          FMSTR_SIZE nResultDataLen);
```

Declaration

```
freemaster.h
```

Implementation

```
freemaster_appcmd.c
```

Arguments

nResultDataAddr (in) - a pointer to data buffer which is to be copied to the Application Command data buffer

nResultDataLen (in) - the length of a data to be copied. It must not exceed the *FMSTR_APPCMD_BUFF_SIZE* value.

Description

This function can be used *before* the Application Command processing is finished, when there are any data to be returned back to the PC.

The response data buffer is copied to the Application Command data buffer, from where it is accessed in case the host requires it. Do not use `FMSTR_GetAppCmdData` and the data buffer, after the `FMSTR_AppCmdSetResponseData` is called.

Note: The current version of the FreeMASTER Tool does not support the Application Command response data.

3.4.5 FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(
          FMSTR_APPCMD_CODE nAppCmdCode,
          FMSTR_PAPPCMDFUNC pCallbackFunc);
```

Declaration

```
freemaster.h
```

Implementation

```
freemaster_appcmd.c
```

Arguments

nAppCmdCode (in) - an Application Command code for which the callback is to be registered

pCallbackFunc (in) - a pointer to a callback function which is to be registered. Use NULL to un-register a callback registered previously with this Application Command.

Return Value

This function returns non-zero value when the callback function was successfully registered or un-registered. It may return zero when you try to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description

This function can be used to register a given function as a callback handler for an Application Command. The Application Command is identified using the single-byte code. The callback function is invoked automatically by the FreeMASTER driver, when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is:

```
FMSTR_APPCMD_RESULT HandlerFunction(
        FMSTR_APPCMD_CODE nAppcmd,
        FMSTR_APPCMD_PDATA pData,
        FMSTR_SIZE nDataLen)
```

Where

nAppcmd - is the Application Command code

pData - points to Application Command data received (if any)

nDataLen - is the information about Application Command data length

The return value of the callback function is used as the Application Command Result Code, and is returned to the FreeMASTER Tool.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or is defined as zero, the `FMSTR_RegisterAppCmdCall` function always fails.

3.5 API Data Types

This section describes the data types used in the FreeMASTER driver. The information provided here may help the user to modify or to port the FreeMASTER Serial Communication Driver to Freescale platforms, which are not yet officially supported. **Please note that the licensing condition prohibits a use of the FreeMASTER tool and FreeMASTER Serial Communication Driver with a non-Freescale microprocessor or microcontroller products.**

Table 3-2 describes the public data types which are used in the FreeMASTER driver API calls. The data types are declared in the freemaster.h header file.

Table 3-2. Public Data Types

Type Name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but may be also a pure integer type. For example, this type is defined as long integer on the 56F8xxx platform where 24bit addresses need to be supported, but the C-pointer may be only 16bit wide in some compiler configuration.
FMSTR_SIZE	Data type used to hold a memory block “size”. It is required that this type is unsigned and at least 16bit wide integer.
FMSTR_BOOL	Data type used as a general “boolean” type. This type is used only in zero/non-zero conditions in the driver code.
FMSTR_APPCMD_CODE	Data type used to hold the Application Command Code. Generally, this is an 8-bit unsigned value.
FMSTR_APPCMD_DATA	Data type used to create Application Command data buffer. Generally, this is an 8-bit unsigned value.
FMSTR_APPCMD_RESULT	Data type used to hold the Application Command Result Code. Generally, this is an 8-bit unsigned value.
FMSTR_PAPPCMDFUNC	Pointer to Application Command handler function. See Section 3.4.5, “FMSTR_RegisterAppCmdCall for more details.

Table 3-3 describes the TSA-specific public data types. These types are declared in the `freemaster_tsa.h` header file, which is included indirectly by the `freemaster.h` to the user application.

Table 3-3. TSA Public Data Types

Type Name	Description
FMSTR_TSA_TINDEX	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables. By default this is defined as FMSTR_SIZE.
FMSTR_TSA_TSIZE	Data type used to hold a memory block size as used in the TSA descriptors. By default this is defined as FMSTR_SIZE.

Table 3-4 describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and are NOT available in the application code.

Table 3-4. Private Data Types

Type Name	Description
FMSTR_U8	The smallest memory entity. On vast majority of platforms, this is the unsigned 8bit integer. On the 56F8xx DSP platform this is defined as unsigned 16 bit integer.
FMSTR_U16	Unsigned 16 bit integer.
FMSTR_U32	Unsigned 32 bit integer.
FMSTR_S8	Signed 8 bit integer. This type is not defined on 56F8xx platform.
FMSTR_S16	Signed 16 bit integer.
FMSTR_S32	Signed 32 bit integer.
FMSTR_FLAGS	Data type making a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8bit wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16bit wide.
FMSTR_BCHR	A single character in a communication buffer. Typically, this is an 8-bit unsigned integer except on DSP platforms where it is a 16 bit integer.
FMSTR_BPTR	A pointer to communication buffer (an array of FMSTR_BCHR).
FMSTR_SCISR	Data type holding the SCI status register value (8, 16 or 32 bit wide depending on the platform).

Chapter 4 PLATFORM-SPECIFIC TOPICS

The following sections describe the implementation details of each supported platform. Please see also the **`readme.txt`** files located in each platform-specific directory in the `src_platforms` directory.

4.1 Platform-dependent Code

Although the FreeMASTER Serial Communication Driver is written in the C language, with cross-platform compatibility and portability in mind, it was necessary to make a small portions of code platform-dependent. The following parts of the code are in the platform-dependent part of the driver:

- **Data types** - code size and performance may be optimized when using data types natively suitable for each particular platform. For example, the FMSTR_SIZE8 data type is only required to be 8-bit wide in the driver code and it is declared as “unsigned char” on most of the platforms. On the other hand, a better code is generated on DSP or Hybrid Controller platforms when this type is declared as 16-bit integer.
- **Communication buffer access and memory access** - typically, the buffers are simple arrays of bytes on most platforms. However, on the 56F8xx DSP platform, nothing like a byte array exists so a native array of 16-bit integers takes the buffer role.
- **16-bit vs. 32-bit FreeMASTER commands** - to optimize the communication protocol traffic, the FreeMASTER protocol defines memory accesses of both the 16-bit and 32-bit wide addresses. Processor platforms differ in the address bus width, so they differ also in how the FreeMASTER driver handles the protocol addresses. On some platforms, it makes sense to implement both modes simultaneously, as there is a kind of “zero-page” RAM which can be addressed using 16bit addresses as well as standard memory requiring 32-bit addressing.
- **SCI handling** - although the SCI modules are fairly similar across the Freescale platforms, the control and status registers may be organized differently and sometimes needs different handling.
- **Interrupt service routine** - Declaration of the interrupt service routine depends heavily on the platform, and on the C compiler used.

Despite the list of platform-dependent exceptions above, the vast majority of the FreeMASTER driver code is common to all platforms. Rarely, some minor platform or compiler dependencies are solved directly in the shared source code files.

- **Driver initialization and serial line handling** - thanks to customized SCI access macros for each platform, the general serial communication handler may be written completely independent on the platform.
- **Protocol decoder and protocol handlers** - the FreeMASTER protocol is completely handled in platform-independent part.
- **Oscilloscope and Recorder data sampling** - thanks to having specialized memory access routines (platform-dependent), the rest of the Oscilloscope and Recorder logic is made completely platform-independent.
- **Target-side Addressing** - all TSA-related code is independent on the target platform.

4.2 DSP56F8xx Digital Signal Processors

The Digital Signal Processors of the 56F800 family was perhaps the most difficult platform to be supported with the FreeMASTER driver and required much code to be moved to the platform-dependent part. The main difficulty is caused by the fact that DSP memory is not organized as bytes, but as 16bit-wide integers, which means there are two bytes on each single memory address. The 56F8xx core does not support the byte arrays and byte-wise access to memory.

4.2.1 56F8xx-specific Driver Files

The 56F8xx-specific code can be found in the **`src_platforms/56F8xx`** directory.

- **`freemaster.h`** - master header file identifies the platform with FMSTR_PLATFORM_56F8xx macro constant
- **`freemaster_56F8xx.h`** - contains the driver options specific to this platform and several other platform-specific memory access inline functions and macros
  - — the 16-bit FreeMASTER protocol commands are only implemented (FMSTR_USE_EX_CMDS is forced to zero)
  - — communication buffer bytes are accessed individually in the separate data words
- **`freemaster_56F8xx.c`** - implements the platform-specific memory access functions

4.2.2 56F8xx-specific Configuration Options

None. Only the standard options are used, as described in Section 2.4, “Driver Configuration.

4.3 56F8xxx Digital Signal Controllers

The Digital Signal Controllers (also referred as Hybrid Controllers) of the 56F8300, 56F8100 and 56F8000 families (also known as 56F800E), combine the DSP-optimized instruction set with a standard microcontroller-like core. Unlike the older 56F8xx family, this platform supports the byte-wise oriented access to memory.

The most of the porting effort was dedicated to enable global memory access (24bit addressing) in the so-called "Small Memory Model" of the CodeWarrior C compiler. In this mode, the compiler assumes the C-pointers are 16-bit wide only, without a concept of "far" addresses (this concept is to be supported in future compiler versions). The "address" in the FreeMASTER driver thus needs to be allocated and accessed as a standard 32-bit wide integer, which requires some inline assembly code when accessing a memory.

The 56F8xxx is the only platform which enables an interrupt-driven real-time communication over the EOnCE/JTAG port. The Real-time Data Exchange feature (RTDX) of the EOnCE port is used in a way similar to how the SCI communicates. The only difference is that JTAG uses 32-bit-wide data as the basic communication entity, while SCI uses a standard byte-oriented communication. With JTAG, each four bytes which would normally be sent over SCI are packed together and transmitted over the JTAG line. Similarly, each double-word received from the JTAG port is split to four separate bytes which are fed to protocol decoding engine.

4.3.1 56F8xxx-specific Driver Files

The 56F8xxx-specific code can be found in the **`src_platforms/56F8xxx`** directory.

- **`freemaster.h`** - master header file identifies the platform with FMSTR_PLATFORM_56F8xxx macro constant
- **`freemaster_56F8xxx.h`** - contains the driver options specific to this platform and several other platform-specific memory access inline functions and macros
  - — using both, the 16-bit and 32-bit FreeMASTER protocol commands are enabled by default (FMSTR_USE_NOEX_CMDS and FMSTR_USE_EX_CMDS are both "one" by default)
  - — inline assembly functions to access memory with "non-pointer" addresses
- **`freemaster_56F8xx.c`** - implements the platform-specific memory access functions and a code needed to use mixed 16-bit and 32-bit protocol commands

4.3.2 56F8xx-specific Configuration Options

Table 4-1 describes the configuration options specific to the 56F8xxx platform, used in addition to the ones described in Section 2.4, "Driver Configuration.

Table 4-1. 56F8xxx Platform

Statement	Values	Description
#define FMSTR_USE_JTAG	boolean (0 or 1)	Define as non-zero when you want to communicate over the JTAG instead of the SCI. Note that a special JTAG communication plug-in for FreeMASTER tool is required on the PC side.
#define FMSTR_USE_JTAG_TXFIX	boolean (0 or 1)	Implements a simple software workaround of the errorneous TDF bit in the 56F8xxx JTAG module. This option should be set in accordance to the FreeMASTER JTAG communication plug-in settings.
#define FMSTR_REC_FARBUFF	boolean (0 or 1)	When recorder is used and the recorder buffer is to be allocated by the driver (FMSTR_REC_OWNBUFF = 0), this option puts the recorder buffer to a section called "fardata". You can then re-code the linker command file of your project to put the buffer to any arbitrary memory address. If the DSP56F800E_Quick_Start environment is used to develop the application, the linker command files are already set in way the "fardata" section is put in external memory, after the address of 0x10000.

4.4 HC08 / HCS08 Microcontrollers

Thanks to the simplicity of the instruction set and a powerful CodeWarrior C compiler, only a minimal effort was needed to port the FreeMASTER driver to the HC08 and HCS08 platforms.

4.4.1 HC08-specific Driver Files

The HC08 and HCS08-specific code can be found in the **`src_platforms/HC08`** directory.

- **`freemaster.h`** - master header file identifies the platform with FMSTR_PLATFORM_HC08 macro constant
- **`freemaster_HC08.h`** - contains the driver options specific to this platform, and several other platform-specific memory access inline functions and macros
  - — the 16-bit FreeMASTER protocol commands are only implemented (FMSTR_USE_EX_CMDS is forced to zero)
  - — as the SCI modules slightly differ on the HC08 and HCS08 families, the SCI access macros are compiled differently in this file (based on the __HCS08__ preprocessor constant)
- **`freemaster_HC08.c`** - implements the platform-specific memory access functions

4.4.2 HC08-specific Configuration Options

Table 4-1 describes the configuration options specific to the HC08 platform, used in addition to the ones described in Section 2.4, “Driver Configuration.

Table 4-2. HC08 Platform

Statement	Values	Description
#define FMSTR_SCI_RX_INTERRUPT	vector number	SCI Receive interrupt vector number. When both RX and TX interrupt vector numbers are defined, the `FMSTR_Isr` is installed automatically as the interrupt service routine for both interrupts (the ”interrupt” keyword is used for the compiler).
#define FMSTR_SCI_TX_INTERRUPT	vector number	SCI Transmit interrupt vector number.

4.5 HC12 / HCS12 / HCS12X Microcontrollers

As supporting only the 16-bit data addressing, the HC12 and HCS12 driver code is very similar to the HC08 code. The 24-bit PPAGE-based "logical" addresses do not have a native support on these platforms, so this addressing mode it is NOT supported in the FreeMASTER driver.

The HCS12X and new GPAGE-based "global" addressing modes enable a transparent access to paged memory, so it is supported also by the FreeMASTER driver. The HCS12X version of the driver also performs an automatic translation from PPAGE- and RPAGE-based logical addresses to global addresses. See FMSTR_LARGE_MODEL configuration option in Table 4-1.

4.5.1 HC12-specific Driver Files

The HC12, HCS12 and HCS12X-specific code can be found in the **`src_platforms/HC12`** directory.

- **`freemaster.h`** - master header file identifies the platform with FMSTR_PLATFORM_HC12 macro constant
- **`freemaster_HC12.h`** - contains the driver options specific to this platform, and several other platform-specific memory access inline functions and macros
  - for HC12 and HCS12 devices, only 16-bit FreeMASTER protocol commands are supported (FMSTR_USE_EX_CMDS is forced to zero)
  - for HCS12X, the 16-bit commands are enabled by default. The 32-bit commands are enabled in "large" data memory models only. Both HCS12X global and logical addresses are correctly handled by the driver.
- **`freemaster_HC12.c`** - implements the platform-specific memory access functions. The HCS12X large model addressing, logical-to-global address translation, and other HCS12X-specific code is also implemented in this file.

4.5.2 HC12-specific Configuration Options

Table 4-1 describes the configuration options specific to the HC12 platform, used in addition to the ones described in Section 2.4, "Driver Configuration.

Table 4-3. HC12 Platform

Statement	Values	Description
#define FMSTR_SCI_INTERRUPT	vector number	SCI interrupt vector number. When defined, the `FMSTR_Isr` is installed automatically as the interrupt service routine for the SCI (the "interrupt" keyword is used for the compiler).
#define FMSTR_LARGE_MODEL	boolean (0 or 1)	**HCS12X only**: Enable support for 24-bit "large" addressing. When enabled, the serial driver decodes properly all kinds of memory addresses: - 16-bit "near" addresses - 23-bit "global" addresses - 24-bit "logical addresses (PAGE + offset) Default: "false" in SMALL or BANKED compiler modes Default: "true" in LARGE compiler mode

4.6 MPC5xx and MPC55xx PowerPC Processors

PowerPC is a true 32-bit platform with a powerful CodeWarrior C compiler. The FreeMASTER driver was ported with a minimum effort.

4.6.1 MPC55xx-specific Driver Files

The MPC55xx-specific code can be found in the **`src_platforms/MPC55xx`** directory.

- **`freemaster.h`** - master header file identifies the platform with FMSTR_PLATFORM_MPC55xx macro constant
- **`freemaster_MPC55xx.h`** - contains the driver options specific to this platform and several other platform-specific memory access inline functions and macros
  — the 32-bit FreeMASTER protocol commands are enabled by default (FMSTR_USE_EX_CMDS defaults to one). A support for 16-bit commands is possible, but it is not enabled by default (FMSTR_USE_NOEX_CMDS defaults to zero)
- **`freemaster_MPC55xx.c`** - implements the platform-specific memory access functions and handles a mixed 16-bit and 32-bit protocol commands if both are used.

4.6.2 MPC5xx-specific Driver Files

Except for the SCI register map and register access macros, the platform dependent part of the driver is identical to the MPC55xx version.

4.6.3 MPC55xx-specific Configuration Options

None. Only the standard options are used, as described in Section 2.4, “Driver Configuration.

4.7 MCF52xx ColdFire Processors

Similarly as with PowerPC processors, the FreeMASTER driver was ported to the ColdFire platform with only a minimum effort.

4.7.1 MCF52xx-specific Driver Files

The MCF52xx-specific code can be found in the **`src_platforms/MCF52xx`** directory.

- **`freemaster.h`** - master header file identifies the platform with FMSTR_PLATFORM_MCF52xx macro constant
- **`freemaster_MCF52xx.h`** - contains the driver options specific to this platform and several other platform-specific memory access inline functions and macros
  - — the 32-bit FreeMASTER protocol commands are enabled by default (FMSTR_USE_EX_CMDS defaults to one). A support for 16-bit commands is possible, but it is not enabled by default (FMSTR_USE_NOEX_CMDS defaults to zero)
- **`freemaster_MCF52xx.c`** - implements the platform-specific memory access functions, and handles a mixed 16-bit and 32-bit protocol commands if both are used.

4.7.2 MCF52xx-specific Configuration Options

None. Only the standard options are used, as described in Section 2.4, “Driver Configuration.

Appendix A References

[1] *DSP56F800 User Manual, DSP56F801-7UM, Freescale Semiconductor*

[2] *56F8300 Peripheral User Manual, MC56F8300UM, Freescale Semiconductor*

[3] *CPU08 Central Processing Unit, CPU08RM/AD, Freescale Semiconductor*

[4] *M68HC12 & HCS12 Microcontrollers, CPU12RM/AD, Freescale Semiconductor*

[5] *MPC5553/5554 Microcontroller Reference Manual, MPC5553/4RM, Freescale Semiconductor*

[6] *MPC555/556 User's Manual, Freescale Semiconductor*

[7] *MPC565/566 User's Manual, Freescale Semiconductor*

Appendix B Revision History

The following revision history table summarizes changes contained in this document.

Revision	Date	Description
1.0	3/2006	Limited initial release
2.0	9/2007	Updated for FreeMASTER version. New Freescale document template used.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

FMSTRSCIDRVUG
Rev. 2.0,
9/2007